KNOWiNK PollPad Plus 1.0 Electronic Poll Book System Source Code Review Test Report for California

KNI-18001SCRTR-01

Prepared for:

Vendor Name	KNOWiNK
Vendor System	PollPad Plus 1.0

Prepared by:

4720 Independence St.
Wheat Ridge, CO 80033
303-422-1566
www.SLICompliance.com

Accredited by the Election Assistance Commission (EAC) for Selected Voting System Test Methods or Services

Revision History

Date	Release	Author	Revision Summary
May 6th, 2018	1.0	*M. Santos*	Initial Release

Disclaimer

The information reported herein must not be used by the client to claim product certification, approval, or endorsement by NVLAP, NIST, or any agency of the Federal Government.

Trademarks

- SLI is a registered trademark of SLI Compliance.
- All products and company names are used for identification purposes only and may be trademarks of their respective owners.

TABLE OF CONTENTS

Overview

Portion of Test Plan Addressed

This Test Report details the Security/Integrity and Vulnerability review of the iOS/Swift and Ruby on Rails source code of the **KNOWiNK PollPad Plus 1.0** electronic poll book system (KNOWiNK System).

References

The following key documents were used in preparing this test plan.

1. California Electronic Poll Book Regulations
2. Swift-Style-Guide
   https://github.com/raywenderlich/swift-style-guide
3. iOS / Swift best practices for 2018
   https://theswiftdev.com/2018/02/09/ios-swift-best-practices-for-2018/
4. Guides.rubyonrails.org
5. Ruby-doc.org
6. https://www.sitepoint.com/10-ruby-on-rails-best-practices

Review Process

Security and Integrity Code Review Process

The iOS/Swift and the Ruby on Rails code bases were reviewed as a manual security and integrity review to analyze the code for findings against the following requirements:

- Adherence to other applicable coding format conventions and standards including best practices for the coding language used, and any IEEE, NIST, ISO, or NSA standards or guidelines which the contractor finds reasonably applicable.
- Evaluation of the use and correct implementation of cryptography and key management.
- Analysis of error and exception handling.
- Evaluation of whether the design and implementation follow sound, generally accepted engineering practices.
- Is code defensively written against:
  - Bad data;
  - Errors in other modules;
  - Changes in environment;

  - User errors; and
  - Other adverse conditions?
- Evaluation of whether the system is designed in a way that allows meaningful analysis, including:
  - Is the architecture and code amenable to an external review (such as this one)?
  - Could code analysis tools be usefully applied?
  - Is the code complexity at a level that it obfuscates its logic?
- Search for embedded, exploitable code (such as “Easter eggs”) that can be triggered to affect the system.

The review process for the code base looked to make the best effort within the time allowed to find and report observations for the above categories. As such, it is understood that there may be undetected vulnerabilities in these categories.

Vulnerability Code Review Process

The iOS/Swift and the Ruby on Rails code bases were reviewed as a manual vulnerability review to analyze the code for findings against the following requirements:

- Search for exposures to commonly exploited vulnerabilities, such as buffer overflows, integer overflow, inappropriate casting, or arithmetic.
- Evaluation of potential vulnerabilities and related issues (code quality and standards compliance), considering that an exploitable issue in a component that is not in itself security relevant could be used to subvert more critical data. This is an issue whenever the architecture of the system does not provide strong separation of the components.
- Analysis of the program logic and branching structure.
- Evaluation of the likelihood of security failures being detected.
  - Are audit mechanisms reliable and tamper resistant?
  - Is data that might be subject to tampering properly validated and authenticated?
- Evaluation of the risk that a user can escalate his or her capabilities beyond those authorized.
- Search for dynamic memory access features which would permit the replacement of certificated executable code or control data or insertion of exploitable code or data.
- Search for use of runtime scripts, instructions, or other control data that can affect the operation of security relevant functions or the integrity of the data.

To the extent possible, reported vulnerabilities include an indication of whether the exploitation of the vulnerability would require access by:

- Voter: Usually has low knowledge of the Electronic Poll Book System's software and/or hardware design and configuration. Some may have more advanced knowledge. May carry out attacks designed by others.
- Poll worker: Usually has low knowledge of the Electronic Poll Book System's software and/or hardware design and configuration. Some may have more advanced knowledge. May carry out attacks designed by others. They have access to the software and/or hardware for up to ten days, but all physical security has been put into place before the machines are received.
- Elections official insider: Wide range of knowledge of the Electronic Poll Book System's software and/or hardware design and configuration. May have unrestricted access for long periods of time. Their designated activities include:
  - Set up and pre-election procedures;
  - Election operation;
  - Post-election procedures; and
  - Archiving and storage operations.
- Vendor insider: With great knowledge of the Electronic Poll Book System's software and/or hardware design and configuration. They have unlimited access to the Electronic Poll Book System's software and/or hardware before it is delivered to the purchaser and, thereafter, may have unrestricted access when performing warranty and maintenance service and when providing election administration services.

SLI does not verify or demonstrate exploitability of the vulnerability.

Any vulnerability theories developed by the source code review team members shall, to the extent possible, be referred to the Secretary of State staff. The review process for the code base looked to make the best effort within the time allowed to find and report observations for the above categories. As such, it is understood that there may be undetected vulnerabilities in these categories.

Review Results

Security and Integrity Source Code Review Analysis

SLI conducted a source code review of the iOS/Swift and Ruby on Rails source code for compliance to the California Electronic Poll Book Regulations

The source code was reviewed for adherence to applicable coding format conventions and standards including best practices for the coding language used, for this review iOS/Swift and Ruby on Rails best practices were reviewed.

- The expected outcome was that no issue would be found.
- The actual outcome was a determination that no issues were found.

The source code was reviewed for evaluation of the use and correct implementation of cryptography and key management

- The expected outcome for this review was that no issue would be found.
- The actual outcome for this review was a determination that no issues were found.

The source code was reviewed for analysis of error and exception handling.

- The expected outcome for this review was that no issue would be found.
- The actual outcome for this review was a determination that no issues were found in the iOS/Swift source code.
- In the Ruby on Rails source code, the actual outcome for this review was a determination that:
  - There are some instances where error handling is not unique to the error or misspelled.
  - There is an inconsistency in the party listings between objects.
  - File/Database open errors are not being checked and leave the possibility for the data to be corrupted or lost

The source code was reviewed for evaluation of whether the design and implementation follow sound, generally accepted engineering practices.

- The expected outcome for this review was that no issue would be found.
- The actual outcome for this review was a determination that no issues were found.

The source code was reviewed for analysis to determine if code is defensively written against:

- Bad data;
- Errors in other modules;
- Changes in environment;
- User errors; and
- Other adverse conditions.
  - The expected outcome for this review was that no issue would be found.
  - In the iOS/Swift source code, the actual outcome for this review was a determination that one issue was found where a Delete function is accessible.
  - In the Ruby on Rails source code, the actual outcome for this review was a determination that this code was defensively written.

The source code was reviewed for evaluation of whether the system is designed in a way that allows meaningful analysis, including:

- Is the architecture and code amenable to an external review (such as this one)?
- Could code analysis tools be usefully applied?
- Is the code complexity at a level that it obfuscates its logic?
  - The expected outcome for this review was that no issue would be found.
  - The actual outcome for this review was a determination that no issues were found.

The source code was reviewed for embedded, exploitable code (such as “Easter eggs”) that can be triggered to affect the system.

- The expected outcome for this review was that no issue would be found.
- The actual outcome for this review was a determination that no issues were found.

Vulnerability Source Code Review Analysis

SLI conducted a Vulnerability source code review of the IOS/Swift and Ruby on Rails source code for compliance to the California Electronic Poll Book Regulations.

The source code was reviewed for exposures to commonly exploited vulnerabilities, such as buffer overflows, integer overflow, inappropriate casting or arithmetic.

- The expected outcome was that no issue would be found.
- In the iOS/Swift source code, the actual outcome for this review, was a determination that a potential memory leak condition was found.

  This vulnerability would require access by a Vendor Insider, or possibly an Elections Official Insider.
- In the Ruby on Rails source code, the actual outcome for this review was a determination that no issues were found.

The source code was reviewed for evaluation of potential vulnerabilities and related issues (code quality and standards compliance), considering that an exploitable issue in a component that is not in itself security relevant could be used to subvert more critical data. This is an issue whenever the architecture of the system does not provide strong separation of the components

- The expected outcome for this review was that no issue would be found.
- In the iOS/Swift source code, the actual outcome for this review was a determination of the following:
  - Reuse of generic error messages could leave the code vulnerable to malicious code without the ability to accurately diagnose and record issues that arise.

  - Improper use of Controller-View-Model programming structure results in the potential for inadvertent data loss.

    These two vulnerabilities would require access by a Vendor Insider.

- In the Ruby on Rails source code, the actual outcome for this review was a determination that no issues were found.

The source code was reviewed for analysis of the program logic and branching structure.

- The expected outcome for this review was that no issue would be found.
- In the iOS/Swift source code, the actual outcome for this review was a determination of the following:
  - Re-use of generic error messages could leave the code vulnerable to malicious code without the ability to accurately diagnose and record issues that arise.
  - Improper use of Controller-View-Model programming structure, potential for inadvertent data loss.

    These two vulnerabilities would require access by a Vendor Insider.
- In the Ruby on Rails source code, the actual outcome for this review was a determination that no issues were found.

The source code was reviewed for evaluation of the likelihood of security failures being detected.

- Are audit mechanisms reliable and tamper resistant?
- Is data that might be subject to tampering properly validated and authenticated?
  - The expected outcome for this review was that no issue would be found.
  - In the iOS/Swift source code, the actual outcome for this review was a determination that the software has the ability to delete log files and that activity is not itself logged, violating standard security practice.

    This vulnerability would require access by a Vendor Insider.
  - In the Ruby on Rails source code, the actual outcome for this review was a determination that four instances of Re-use of Generic Error messages could leave the code vulnerable to malicious errors were found.

    Either a Vendor Insider or Election Official Insider may be able to manipulate the error messaging.

The source code was reviewed for evaluation of the risk that a user can escalate his or her capabilities beyond those authorized

- The expected outcome for this review was that no issue would be found.

- The actual outcome for this review was a determination that no issues were found.

The source code was reviewed to evaluate for dynamic memory access features which would permit the replacement of certificated executable code or control data or insertion of exploitable code or data

- The expected outcome for this review was that no issue would be found.
- In the iOS/Swift source code, the actual outcome for this review was a determination that no issues were found.
- In the Ruby on Rails source code, the actual outcome for this review was a determination that use of sidekiq and sideqit to handle data queues have the potential for stack overflow if not managed properly.

  The Vendor Insider may be able to alter the outcome should data be lost.

The source code was reviewed for use of runtime scripts, instructions, or other control data that can affect the operation of security relevant functions or the integrity of the data.

- The expected outcome for this review was that no issue would be found.
- The actual outcome for this review, in the iOS/Swift source code, was a determination as follows:
  - General comment – Severe errors should log (not display) traceback for forensic purposes (e.g. i.backtrace.inspect) for all fatal conditions.
  - Case statements should utilize default path in all cases.

    The Vendor Insider may be able to alter the outcome should data be lost.
- In the Ruby on Rails source code, the actual outcome for this review was a determination that no issues were found.

Findings

This section summarizes any Findings from the **KNOWiNK PollPad Plus 1.0** Security and Integrity, iOS/Swift and Ruby on Rails source code review.

Security and Integrity Source Code Review Discrepancies

iOS/Swift

In the iOS/Swift source code review analysis to determine if code is defensively written showed that one issue was found where a Delete function is accessible.

Ruby on Rails

In the Ruby on Rails source code review, analysis of error and exception handling showed:

- There are some instances where error handling is not unique to the error or misspelled.
- There is an inconsistency in the party listings between objects.
- File/Database open errors are not being checked and leave the possibility for the data to be corrupted or lost

Vulnerability Source Code Review Discrepancies

iOS/Swift

In the iOS/Swift source code review:

- A potential memory leak condition was found.

  This vulnerability would require access by a Vendor Insider, or possibly an Elections Official Insider.
- Re-use of generic error messages could leave the code vulnerable to malicious code without the ability to accurately diagnose and record issues that arise.
- Improper use of Controller-View-Model programming structure, potential for inadvertent data loss.

  These two vulnerabilities would require access by a Vendor Insider.
- The software has the ability to delete log files and that activity is not itself logged, violating standard security practice.

  This vulnerability would require access by a Vendor Insider.
- General comment – Severe errors should log (not display) traceback for forensic purposes (e.g. i.backtrace.inspect) for all fatal conditions.
- Case statements should utilize default path in all cases.

  The Vendor Insider may be able to alter the outcome should data be lost

Ruby on Rails

In the Ruby on Rails source code review:

- Four instances of Re-use of Generic Error messages could leave the code vulnerable to malicious errors were found.

  Either the Vendor Insider or Election Official Insider may be able to manipulate the error messaging.
- The use of sidekiq and sideqit to handle data queues have the potential for stack overflow if not managed properly.

  The Vendor Insider may be able to alter the outcome should data be lost.

Conclusion

For the Security and Integrity review, four Findings were located within the **KNOWiNK PollPad Plus 1.0** electronic poll book system code base.

In the Vulnerability review, 11 Findings were located within the **KNOWiNK PollPad Plus 1.0** electronic poll book system code base.